Introduction

P. Calafiura

Event Data Model mini-Workshop

July 11, 2000

Data Model Integration (from May Workshop)

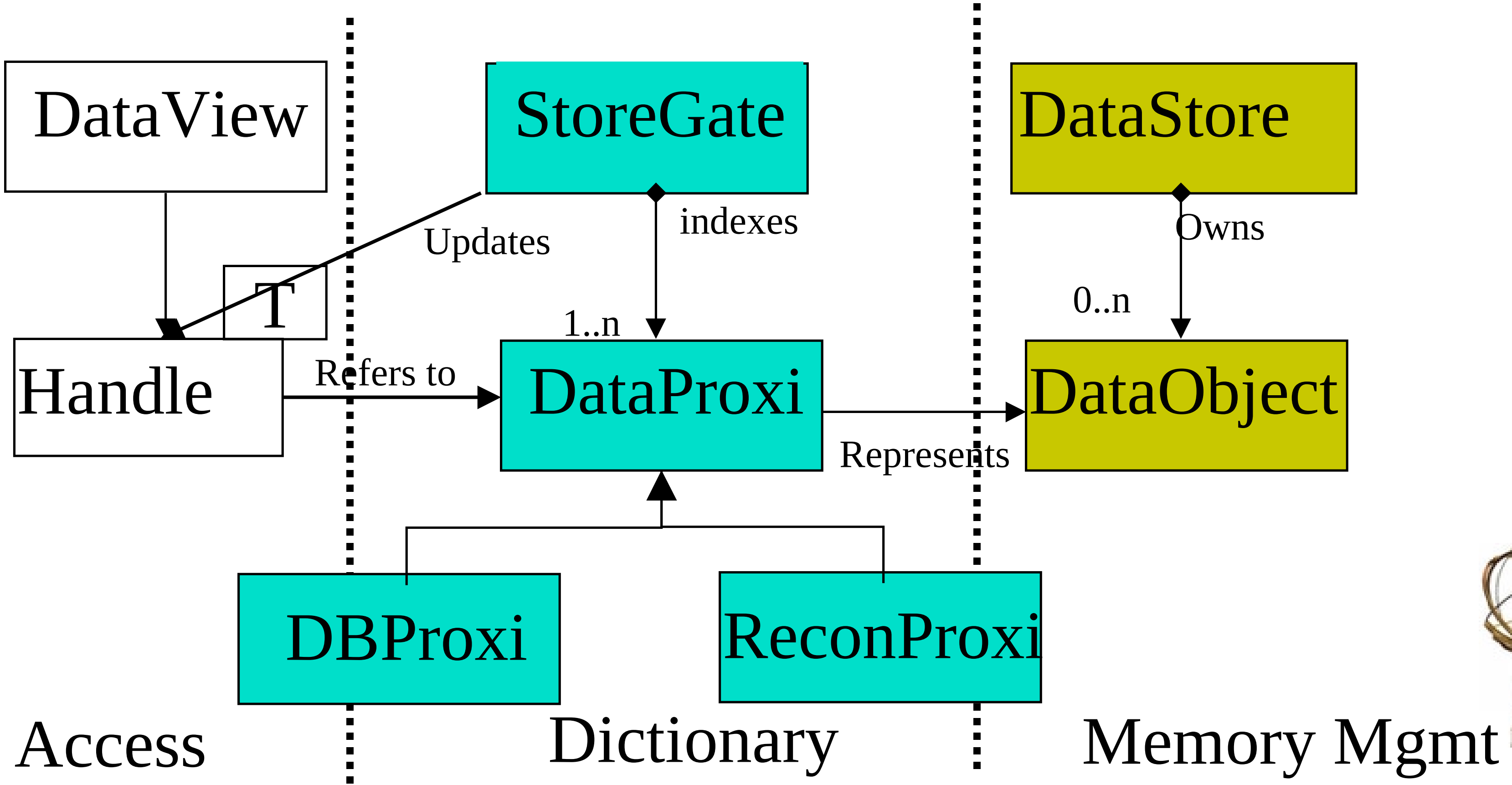
- We more or less agree on
  - **Helpers to support multiple logical "views"**
  - **Typed access (compile or run-time)**
  - **WORM store: can only add to it**
    - **ATLFast annotations, extend collections**
  - **DataObject relationships: no "forward pointers"**
- **We have to converge on**
  - **What is returned? STL-like iterator, Handle, plain C++ pointer/ref**
  - **How do modules define what they want?**
    - **(Default) Keys, Selectors**

Data Models side-by-side

	BaBar	D0	Gaudi
Data Obj	-	Chunk<Coll>	DataObject
Inter-Obj Rel	Proxy	LinkIndex LinkPtr<T>	linkID SmartRef<T>
Key	AbsKey	TKey	string
Handle	?	THandle	~SmartDataPtr
coll/iter	-	Chunk Selector	ObjVector
Trans/Pers	ProxyDict	d0Ref	Opaque Addr/CnvSvc
directory	-	-	IDataDir

Views, Proxies and the TDS

- **View: client view of the stores, updated by the stores**
- **StoreGate: type-safe store access, implements cache policy**
- **Handle: smart ptr & iterators, basic client interface**
- **DataProxi: access control, build the DataObject on demand**
- **DBProxi, ReconProxi, …: concrete DataProxies**



StoreGate Prototype *

- **Focus on Interface. Use Gaudi TDS to implement it**

- **Key: optional, distinguish data objects of same type**

```
Identifier id = at_id.lar_em();
LArCellContainer::Key key(id);
```

- **Selector: optional, selection based on DataObject content**

```
LArCellSelector* sel = new LArCellSelector(100);
```

- **THandle: smart pointer, provide iterator access as well**

```
THandle<LArCellContainer> myhandle(sel);
```

- **StoreGateSvc: type-safe access to Gaudi TDS**

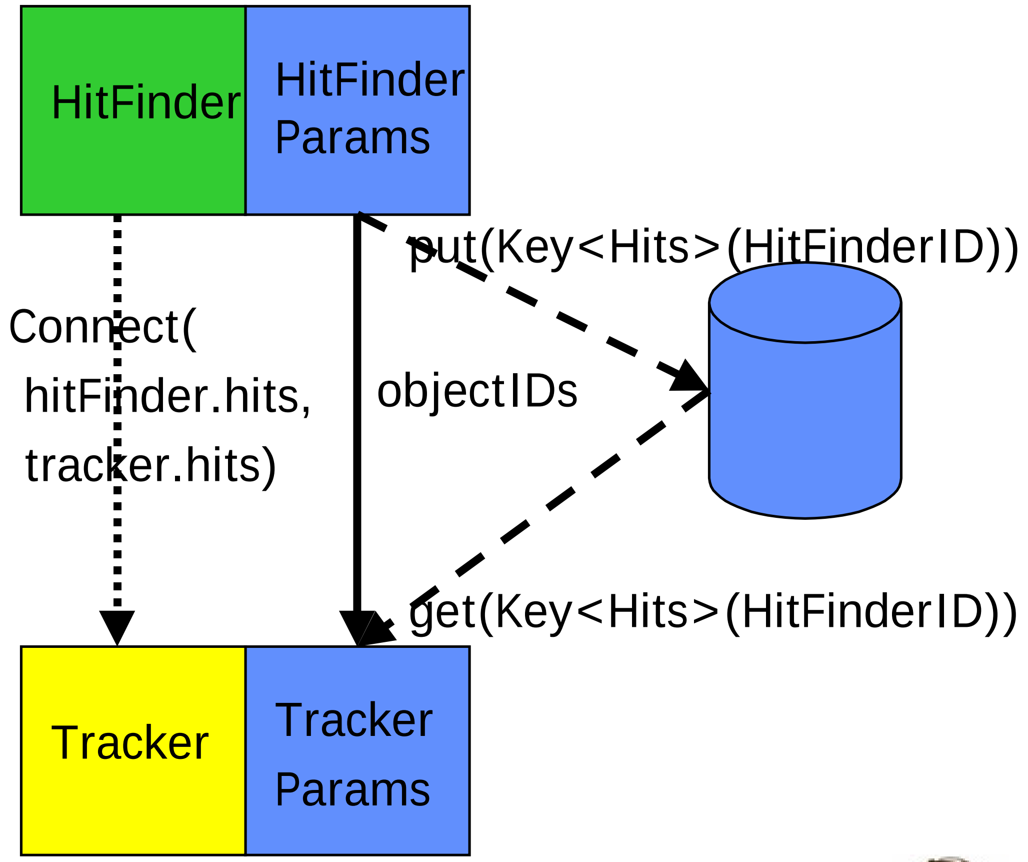
```
StatusCode sc =
  storeGateSvc()->retrieveObject(key, myhandle);
cout << ”No of Cells=" << myhandle->size() << endl;
LArCellContainer::const_iterator first =
      myhandle->begin();
LArCellContainer::const_iterator last =
      myhandle->end();
for (; first != last; ++first)
   float energy= (*first)->energy();
```

Locating DataObjects in TES

- price to pay for reduced physical coupling
- strategies
  - —string in jobOptions
    - simple, will it scale?
  - —use Key class
    - compiler helps checking, but still need jobOptions
  - —Identify a DataObj using its type
    - what if more than one (use Selectors)
    - derived types!
  - —Identify a DataObj using its maker
    - “connect” sintax, Object Networks, I/O ports

Simulated Data Flow (old stuff) *

- Identify Algos input and output DataObj
- Hide TES details
  - —like a view
- Connect them using "directives"
- Could also use as marshalling layer for multi-language


HitFinder
HitFinder Params
put(Key<Hits>(HitFinderID))
Connect(hitFinder.hits, tracker.hits)
objectIDs
get(Key<Hits>(HitFinderID))
Tracker
Tracker Params

Inter-Object Relationships

- No concrete design proposal yet

- use same access method as for direct Store retrieval (e.g. Handle)
- reduce Disk/tape access (lazy evaluation)
- allow to cross technology boundaries?
  - my ROOT nano-DST points to an Objectivity collection
- use association classes (HitsOnTrack) for relationships pointing “forward in time”

Next

- Srini: Discussion and Prototype Design for TES Access
- Hong: StoreGate Prototype